المولف المبرمج الروسي Ms-Rem

الموقع الاصلي

http://wasm.ru/article.php?article=hiddndt

اسم المقاله العثور على الـlatent processes

اعتاد الكثيرمن مستخدمي الويندوز على ان ال Task manager يقوم بعرض جميع الـprocesses . ويعتقد الكثيرون ان الاختفاء عن انظار الـTask manage امر مستحيل في حين ان ذلك امر سهل جدا.لفعل ذلك هناك العديد من الاساليب وتنفيذ ذلك امر سهل المنال !!! يبقى فقط سؤال وحيد لماذا تندر الـtrojan التي تستخدم هذا الاسلوب ؟في الحقيقة هذا النوع يشكل فقط حوالي 1000/1 !! الى نسبى الـtrojan التي لا تجيد الاختفاء !!!
اعتقد ان السبب هو ان كون صانعي هذه الـtrojan (الكسالي) لا يكتبون احصنتهم الخاصى بانفسهم وانما ياخذون برامج جاهزة ويضيفون اليها البرامج الخاصة بهم مما يعني ان الـtrojan الخفية ستنتشر بدورها عما قريب

من الطبيعي اننا نحتاج الى حما ية من هذه الاشياء. في الواقع ان منتجي مضادات الفيروسات والجدران النارية اصبحوا متخلفين (الى حد ما) بما ان منتجاتهم لا تستطيع العثور على الـlatent processes . القليل منها يفعل ذلك وواحد فقط مجاني Klister و(هو لا يعمل الا على win2000) اما البقية فهي غالية الثمن مع انه تكلف القليل من المال والجهد!!!

كل البرامج التي الموجودة الان لايجاد الـlatent processes تقوم على مبدا معين لذلك بامكاننا ان نفكرفي طريقة معينة للالتفاف حول هذا المبدا او ان يلحق الفيروس نفسه ببرنامج معين وهذا هو الاسهل عند التنفيذ . ان المستخدم الذي يشتري برنامجا تجاريا لا يستطيع تغييره لذلك الحاق الفيروس ببرنامج معين تعتبر وسيلة فعاله وهذا هو الاسلوب المستخدم في الـRootKits
التجارية على سبيل المثال (hxdef Golden edition) .
الحل الوحيد هو انشاء برنامج مجاني مفتوح المصدر للعثور على الـlatent processes والذي يطبق العديد من مبادي‌ء اظهار الـlatent processes وهذا سيكفل لنا الحماية (على الاقل) من الفيروسات التي تستخدم الاساليب الاساسية للاخفاء اما عن الـlatent processes الذي يلحق نفسه ببرنامج معين فبامكان المستخدم ان اخذ الكود وتعديله بحيث يناسب متطلباته

في هذا المقال اريد ان اناقش الاساليب الاساسية المتبعه في للعثور على latent processes ,كمااريد ان اضع امثلة لاكواد تنفذ هذه الاساليب . واخيرا انشاء برنامج يقوم بتحقيق المطالب المذكورة اعلاه

Detection in User Mode

في البدايه سنناقش الاساليب البسيطة المستخدمة في البحث والتي يمكن ان تطبق في 3 حلقات بدون الحاجة الى استخدام الـDrivers.
المبدا الذي تقوم عليه هو كون كل started process يملك اثار جانبية ناتجة عن عمله يمكن بواسطتها العثور عليه مثلا: ال handles و النوافذ و System objects (هذا هو سبب تسميتها User Mode المترجم). اذا كانت هذه هي الطرق المستخدمة فان الاختفاء عن اعين البارامج التي تستخدم هذه الطريقة امر يسير ولكن لذلك لابد من الاخذ بعين الاعتبار كل الاثار الجانبية للـprocess وهذا لم ينفذ الى الان ولا في اي من Rootkit المنشورة الى الان.
حقيقة هذه الاساليب حقيقة سهلة في التنفيذ- امنه عند التطبيق –تعطي نتائج فعالة لذلك من الافضل ان ناخذها في الحسبان.

سوف نتفق على صيغة النتائج التي ستعيدها دوال البحث وليكن هذا عبارة عن linked list

type

```
 PProcList = ^TProcList;
 TProcList = packed record
   NextItem: pointer;
   ProcName: array [0..MAX_PATH] of Char;
   ProcId: dword;
  ParrentId: dword;
 end;
```

الحصول على قائمة بال processes عن طريق ToolHelp API

في البداية سوف نعرف داله نموذجية والتي ستحصل على قائمة بالprocesses وسنقارن نتائجها بالنتائج التي سنحصل عليها بواسطة الطرق الاخرى

```
{
 Получение списка процессов через ToolHelp API.
```

الحصول على قائمة بال processes عن طريق ToolHelp API

```
}
procedure GetToolHelpProcessList(var List: PListStruct);
var
 Snap: dword;
 Process: TPROCESSENTRY32;
 NewItem: PProcessRecord;
begin
  Snap := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if Snap <> INVALID_HANDLE_VALUE then
     begin
      Process.dwSize := SizeOf(TPROCESSENTRY32);
      if Process32First(Snap, Process) then
         repeat
          GetMem(NewItem, SizeOf(TProcessRecord));
          ZeroMemory(NewItem, SizeOf(TProcessRecord));
          NewItem^.ProcessId  := Process.th32ProcessID;
          NewItem^.ParrentPID := Process.th32ParentProcessID;
          lstrcpy(@NewItem^.ProcessName, Process.szExeFile);
          AddItem(List, NewItem);
         until not Process32Next(Snap, Process);
      CloseHandle(Snap);
     end;
end;
```

من الطببعي اننا لن نستطيع العثور على اي latent process بهذه الطريقة لهذا فان هذه القائمة ستكون نموذجا للتفريق بين ال latent processes و غيرها .

الحصول على قائمة بال processes عن طريق Native API

المستوى الثاني في الاختبار هو الحصول على قائمة بال processes عن طريق (Native API)
ورغم انه من غير المحتمل الحصول على نتائج ولكن لا باس من المحاولة

```
{
  Получение списка процессов через ZwQuerySystemInformation.
```

الحصول على قائمة بالprocesses عن طريق ZwQuerySystemInformation

```
}
procedure GetNativeProcessList(var List: PListStruct);
var
 Info: PSYSTEM_PROCESSES;
 NewItem: PProcessRecord;
 Mem: pointer;
begin
  Info := GetInfoTable(SystemProcessesAndThreadsInformation);
  Mem := Info;
  if Info = nil then Exit;
  repeat
   GetMem(NewItem, SizeOf(TProcessRecord));
   ZeroMemory(NewItem, SizeOf(TProcessRecord));
   lstrcpy(@NewItem^.ProcessName,
           PChar(WideCharToString(Info^.ProcessName.Buffer)));
   NewItem^.ProcessId  := Info^.ProcessId;
   NewItem^.ParrentPID := Info^.InheritedFromProcessId;
   AddItem(List, NewItem);
   Info := pointer(dword(info) + info^.NextEntryDelta);
  until Info^.NextEntryDelta = 0;
  VirtualFree(Mem, 0, MEM_RELEASE);
end;
```

الحصول على قائمة بال processes عن طريق الhandles opened

العديد من البرامج تخفي الprocesses التابعة لها ولا تخفي الhandles لذلك من الممكن انشاء قائمة بالhandles ثم تبعا لذلك قائمة بالprocesses

```
{
  Получение списка процессов по списку открытых хэндлов.
  Возвращает только ProcessId.
}
procedure GetHandlesProcessList(var List: PListStruct);
var
 Info: PSYSTEM_HANDLE_INFORMATION_EX;
 NewItem: PProcessRecord;
 r: dword;
 OldPid: dword;
begin
  OldPid := 0;
  Info := GetInfoTable(SystemHandleInformation);
  if Info = nil then Exit;
  for r := 0 to Info^.NumberOfHandles do
    if Info^.Information[r].ProcessId <> OldPid then
     begin
       OldPid := Info^.Information[r].ProcessId;
       GetMem(NewItem, SizeOf(TProcessRecord));
       ZeroMemory(NewItem, SizeOf(TProcessRecord));
       NewItem^.ProcessId   := OldPid;
       AddItem(List, NewItem);
     end;
  VirtualFree(Info, 0, MEM_RELEASE);
end;
```

في هذا المستوى قد نعثر على شي ولكن لاينصح الاكتفاء بنتائج هذا البحث لان اخفاء الhandle ليس اصعب من اخفاء ال Process كل ما في الامر ان العديد ينسون هذا الامر

الحصول على قائمة بال processes عن طريق الwindows opened

ان الحصول على قائمة بال Registered windows في النظام ثم استدعاء الGetWindowThreadProcessId من الممكن انشاء قائمة بال processes

```
procedure GetWindowsProcessList(var List: PListStruct);

 function EnumWindowsProc(hwnd: dword; PList: PPListStruct): bool; stdcall;
 var
  ProcId: dword;
  NewItem: PProcessRecord;
 begin
  GetWindowThreadProcessId(hwnd, ProcId);
   if not IsPidAdded(PList^, ProcId) then
    begin
     GetMem(NewItem, SizeOf(TProcessRecord));
     ZeroMemory(NewItem, SizeOf(TProcessRecord));
     NewItem^.ProcessId   := ProcId;
     AddItem(PList^, NewItem);
  end;
  Result := true;
 end;

begin
 EnumWindows(@EnumWindowsProc, dword(@List));
end;
```

الحصول على قائمة بال processes عن طريق Direct system call

بمان غالبية الprocesses في الUser Mode تعمل غلغلة ذاتها في processes اخر او عن طريق التقاط الداله ZwQuerySystemInformation من الملف ntdll.dll
في الحقيقة ان دوال الملف ntdll.dll هي عبارة عن system call interface للتعامل مع نواة النظام مثل ال(Int 2EH في Win2000) و (sysenter في XP) لذلك فان اسهل واكثر الطرق فعالية في العثور على ال latent process هو التعامل المباشر مع system call interface مع تغيير الAPI !!!

ان النسخة المعدلة من الداله ZwQuerySystemInformation ستبدو بهذا الشكل في widows XP :

```
{
System call (ZwQuerySystemInformation) for windows XP

}
Function XpZwQuerySystemInfoCall(ASystemInformationClass: dword;
                                 ASystemInformation: Pointer;
                                 ASystemInformationLength: dword;
                                 AReturnLength: pdword): dword; stdcall;
asm
 pop ebp
 mov eax, $AD
 call @SystemCall
 ret $10
 @SystemCall:
 mov edx, esp
 sysenter
end;
```

اما في windows 2000 فان الكود السابق سيبدو هكذا:

```
{
  Системный вызов ZwQuerySystemInformation для Windows 2000.
```

```
System call (ZwQuerySystemInformation) for windows 2000

}
Function Win2kZwQuerySystemInfoCall(ASystemInformationClass: dword;
                                    ASystemInformation: Pointer;
                                    ASystemInformationLength: dword;
                                    AReturnLength: pdword): dword; stdcall;
asm
 pop ebp
 mov eax, $97
 lea edx, [esp + $04]
 int $2E
 ret $10
end;
```

والان لم يتبقى الا ان نعدد الprocesses ليس بالداله الماخوذذة من الملف ntdll.dll وانما بالداله تي عرفناها قبل قليل :

```
procedure GetSyscallProcessList(var List: PListStruct);
var
 Info: PSYSTEM_PROCESSES;
 NewItem: PProcessRecord;
 mPtr: pointer;
 mSize: dword;
 St: NTStatus;
begin
 mSize := $4000;
 repeat
  GetMem(mPtr, mSize);
  St := ZwQuerySystemInfoCall(SystemProcessesAndThreadsInformation,
                              mPtr, mSize, nil);
  if St = STATUS_INFO_LENGTH_MISMATCH then
    begin
      FreeMem(mPtr);
      mSize := mSize * 2;
    end;
 until St <> STATUS_INFO_LENGTH_MISMATCH;
 if St = STATUS_SUCCESS then
  begin
    Info := mPtr;
    repeat
     GetMem(NewItem, SizeOf(TProcessRecord));
     ZeroMemory(NewItem, SizeOf(TProcessRecord));
     lstrcpy(@NewItem^.ProcessName,
             PChar(WideCharToString(Info^.ProcessName.Buffer)));
     NewItem^.ProcessId  := Info^.ProcessId;
     NewItem^.ParrentPID := Info^.InheritedFromProcessId;
     Info := pointer(dword(info) + info^.NextEntryDelta);
     AddItem(List, NewItem);
    until Info^.NextEntryDelta = 0;
  end;
 FreeMem(mPtr);
end;
```

هذا الاسلوب في الواقع سيعثر على كل ال root kits مثلا كل اصدارات الhxdef يمكن العثور عليها بهذه الطريقة

الحصول على قائمة بالprocesses عن طريق تحليل الhandles المتصلة بها

يمكن ان نذكر طريقة اخرى تعتمد هي ايضا علىسرد الhandles .

فكرة هذا الطريقة هي كالتالي : لن نبحث عن الhandles التي تتبع الprocess (الذي نريد العثور عليه) وانما سوف نبحث عنhandles تخص process اخر بحيث يكون الprocess الاخر متصلا بالprocess الذي نبحث عنه .قد تكون الhandles تابعة لنفس الprocess او تابعة لاحد الthreads . في حاله العثور على handle process فان تحديد الPID يمكن طريق الدالة ZwQueryInformationProcess اما اذا كان الامر يخص thread فاننا سنستخدم ZwQueryInformationThread ومنها نستطيع الحصول على الid الذي يتبعه هذا الthread
ان جميع الprocesses التي تعمل في وقتما تم تشغيلها من قبل process ما, اذا فان الprocess الاب يملك جميع الhandles التابعة لابناءه طبعا اذا (هذااذا لم يغلق الprocess الابن) كما ان الhandles التابعة لجميع ال process النشطة موجودة في ال(csrss.exe) client/server run-time subsystem
لاحظوا انه في windows NT تستخدم الJob objects بشكل كبير والتي تسمح بدمج الprocess مثلا (كل الprocesses التايعة لسمتخدم معين او مثلا كل الprocesses التي تؤدي مهمة معينة)
لذلك اذا وجدت الhandle التابع لJob فمن المفيد جدا ان لا تهمل فرصه الحصول على كل الid التي يوحدها هذا الjob

```
{
Получение списка процессов через проверку хэнжлов в других процессах.
                 other processes handles عن طريق processesالحصول على قائمة بال
}
procedure GetProcessesFromHandles(var List: PListStruct; Processes, Jobs,
Threads: boolean);
var
 HandlesInfo: PSYSTEM_HANDLE_INFORMATION_EX;
 ProcessInfo: PROCESS_BASIC_INFORMATION;
 hProcess : dword;
 tHandle: dword;
 r, l     : integer;
 NewItem: PProcessRecord;
 Info: PJOBOBJECT_BASIC_PROCESS_ID_LIST;
 Size: dword;
 THRInfo: THREAD_BASIC_INFORMATION;
begin
 HandlesInfo := GetInfoTable(SystemHandleInformation);
 if HandlesInfo <> nil then
 for r := 0 to HandlesInfo^.NumberOfHandles do
   if HandlesInfo^.Information[r].ObjectTypeNumber in [OB_TYPE_PROCESS,
OB_TYPE_JOB, OB_TYPE_THREAD] then
    begin
      hProcess  := OpenProcess(PROCESS_DUP_HANDLE, false,
                         HandlesInfo^.Information[r].ProcessId);

      if DuplicateHandle(hProcess, HandlesInfo^.Information[r].Handle,
                         INVALID_HANDLE_VALUE, @tHandle, 0, false,
                         DUPLICATE_SAME_ACCESS) then
            begin
             case HandlesInfo^.Information[r].ObjectTypeNumber of
               OB_TYPE_PROCESS : begin
                     if Processes and (HandlesInfo^.Information[r].ProcessId =
CsrPid) then
                     if ZwQueryInformationProcess(tHandle,
ProcessBasicInformation,
                                             @ProcessInfo,
                                             SizeOf(PROCESS_BASIC_INFORMATION),
                                             nil) = STATUS_SUCCESS then
                     if not IsPidAdded(List, ProcessInfo.UniqueProcessId) then
                        begin
                        GetMem(NewItem, SizeOf(TProcessRecord));
                        ZeroMemory(NewItem, SizeOf(TProcessRecord));
                        NewItem^.ProcessId   := ProcessInfo.UniqueProcessId;
                        NewItem^.ParrentPID  :=
ProcessInfo.InheritedFromUniqueProcessId;
                        AddItem(List, NewItem);
                        end;
                     end;
```

```
                OB_TYPE_JOB      : begin
                                   if Jobs then
                                    begin
                                     Size :=
SizeOf(JOBOBJECT_BASIC_PROCESS_ID_LIST) + 4 * 1000;
                                     GetMem(Info, Size);
                                     Info^.NumberOfAssignedProcesses := 1000;
                                     if QueryInformationJobObject(tHandle,
JobObjectBasicProcessIdList,
                                                                  Info, Size,
nil) then
                                        for l := 0 to
Info^.NumberOfProcessIdsInList - 1 do
                                          if not IsPidAdded(List,
Info^.ProcessIdList[l]) then
                                            begin
                                             GetMem(NewItem,
SizeOf(TProcessRecord));

                                             ZeroMemory(NewItem,
SizeOf(TProcessRecord));

                                             NewItem^.ProcessId   :=
Info^.ProcessIdList[l];

                                             AddItem(List, NewItem);
                                            end;
                                     FreeMem(Info);
                                    end;
                                   end;

                OB_TYPE_THREAD   : begin
                                   if Threads then
                                   if ZwQueryInformationThread(tHandle,
THREAD_BASIC_INFO,
                                                               @THRInfo,

SizeOf(THREAD_BASIC_INFORMATION),
                                                               nil) =
STATUS_SUCCESS then
                                     if not IsPidAdded(List,
THRInfo.ClientId.UniqueProcess) then
                                      begin
                                        GetMem(NewItem, SizeOf(TProcessRecord));
                                        ZeroMemory(NewItem,
SizeOf(TProcessRecord));

                                        NewItem^.ProcessId   :=
THRInfo.ClientId.UniqueProcess;

                                        AddItem(List, NewItem);
                                      end;
                                  end;

              end;
              CloseHandle(tHandle);
             end;
           CloseHandle(hProcess);
         end;
 VirtualFree(HandlesInfo, 0, MEM_RELEASE)
;end;
```

للاسف الشديد فان الطرق المذكورة اعلاه لا تسمح بالحصول على اسم الprocess وانما تعطينا فقط الPID لذلك لابد من اجادة الحصول على اسم الprocess بواسطة الPID وارجو ان الايتبادر الTool Help API الى ذهنك بما ان الprocess قد يكون خفيا لذلك سوف نلجا الى فتح ذاكرة الprocess للقراءة وسوف نقرا اسمه من الPEB

طبعا عنوان الPEB من الممكن الحصول عليه بواسطة الداله ZwQueryInformationProcess.

اليكم الكود الذي سيفعل ذلك

```
function GetNameByPid(Pid: dword): string;
var
```

```
 hProcess, Bytes: dword;
 Info: PROCESS_BASIC_INFORMATION;
 ProcessParametres: pointer;
 ImagePath: TUnicodeString;
 ImgPath: array[0..MAX_PATH] of WideChar;
begin
 Result := '';
 ZeroMemory(@ImgPath, MAX_PATH * SizeOf(WideChar));
 hProcess := OpenProcess(PROCESS_QUERY_INFORMATION or PROCESS_VM_READ, false,
Pid);
 if ZwQueryInformationProcess(hProcess, ProcessBasicInformation, @Info,
                              SizeOf(PROCESS_BASIC_INFORMATION), nil) =
STATUS_SUCCESS then
  begin
   if ReadProcessMemory(hProcess, pointer(dword(Info.PebBaseAddress) + $10),
                        @ProcessParametres, SizeOf(pointer), Bytes) and
      ReadProcessMemory(hProcess, pointer(dword(ProcessParametres) + $38),
                        @ImagePath, SizeOf(TUnicodeString), Bytes)  and
      ReadProcessMemory(hProcess, ImagePath.Buffer, @ImgPath,
                        ImagePath.Length, Bytes) then
        begin
          Result := ExtractFileName(WideCharToString(ImgPath));
        end;
   end;
 CloseHandle(hProcess);
end;
```

طبعا رحلة البحث عن الlatent process لم تنتهي بهذا الامر بالامكان التفكير في في طريقة اخرى اذا فك بذلنا لعض الجهد مثلا ان نحقن dll معين في بواسطة الدالة ال`SetWindowsHookEx` والذي يليه تحليل قائمة الprocesses التي تحتوي على our dll لكننا سنكتفي بهذا القدر من الاساليب. طبعا الميزة الاساسية هي سهولة برمجة هذا الاساليب لكع عيبها هو انها تكشف فقط الlatent process التي تعتمد على التقاط الAPI في الUser mode

Kernel Mode detection

اخبر وصلنا الى طرق ايجاد الlatent processes في الKernel Mode . تتميز هذه الاساليب عن غيرها بامكانية تكوين قائمة بالprocesses دون اللجوء الى الAPI وانما بالتعامل مباشرة مع الTask Scheduler . ان الاختفاء عن انظار هذه الطرق امر في غاية الصعوبة لانها هذه الاساليب تعمل بنفس مبادئ عمل نظام التشغيل وان حذف جميع الاثار التي يخلفها الprocess ستؤدي حتما الى توقفه عن العمل

اذا ماهو الprocess ؟ مما يتكون ؟
طبعا لكل process عنوانه الخاص في الذاكرة وhandle و threads و. وهذا يتصل اتصالا وثيق بالبنية الداخلية لنواة النظام

ان المواصفات الخاصة بprocess توجد في structure من النوع EPROCESS وكل الstructures الخاصة بجميع الprocess تتصل مع بعضها البعض بواسطة حلقة مكونة من linked listed
احد الطرق المستخدمة لاخفاء الprocess هي تغيير موضع المؤشرات بحيث لايدخل في القائمة عند المرور بجميع عناصر الحلقة طبعا لكي يعمل الprocess فليس من الضروري ان يدخل في قائمة الprocesses (التي نكونها بالمرور على جميع عناصر الحلقة) لكن من الضروري وجوده في هذه الحلقة لكي يعمل .
طبعا كل الاساليب المتبعة في ايجاد الlatent process متصلة بشكل او باخر بالعثور على الEPROCESS strctute
في البداية سنتفق على صيغة البيانات التي سنحصل عليهاعن ال الlatent process ولابد ان تكون هذه الصيغة مريحة لنقل البيانات من المصدر الى برنامجنا ولتكن هذه الصيغة بالشكل التالي

```
typedef struct _ProcessRecord
{
	ULONG       Visibles;
	ULONG       SignalState;
    BOOLEAN     Present;
	ULONG       ProcessId;
	ULONG       ParrentPID;
	PEPROCESS   pEPROCESS;
	CHAR        ProcessName[256];
} TProcessRecord, *PProcessRecord;
```

ولنتفترض ان هذه الstructure مرتبة على شكل قائمة وليكن من اجل اخر عنصر فيها الحقل present مساويا 0

الحصول على قائمة بالprocesses عن طريق الZwQuerySystemInformation في النواة

```
PVOID GetNativeProcessList(ULONG *MemSize)
{
	ULONG PsCount = 0;
	PVOID Info = GetInfoTable(SystemProcessesAndThreadsInformation);
	PSYSTEM_PROCESSES Proc;
	PVOID Mem = NULL;
	PProcessRecord Data;

	if (!Info) return NULL; else Proc = Info;

	do
	{
		Proc = (PSYSTEM_PROCESSES)((ULONG)Proc + Proc->NextEntryDelta);
		PsCount++;
	} while (Proc->NextEntryDelta);

	*MemSize = (PsCount + 1) * sizeof(TProcessRecord);

	Mem = ExAllocatePool(PagedPool, *MemSize);

	if (!Mem) return NULL; else Data = Mem;

	Proc = Info;
	do
	{
		Proc = (PSYSTEM_PROCESSES)((ULONG)Proc + Proc->NextEntryDelta);
		wcstombs(Data->ProcessName, Proc->ProcessName.Buffer, 255);
		Data->Present    = TRUE;
		Data->ProcessId  = Proc->ProcessId;
		Data->ParrentPID = Proc->InheritedFromProcessId;
		PsLookupProcessByProcessId((HANDLE)Proc->ProcessId, &Data-
>pEPROCESS);
		ObDereferenceObject(Data->pEPROCESS);
		Data++;
	} while (Proc->NextEntryDelta);

	Data->Present = FALSE;

	ExFreePool(Info);

	return Mem;
}
```

لنفترض ان هذه الدالة ستكون الداله النموذجية بما اننا لن نستطيع العثور على اي latent process في الkernel mode

سنستخدم GetInfoTable فقط من اجل الحصول على معلومات لمن لا يعرف ما اعنيه سقت هذا الكود لتبيين المقصود .

```
/*
  Получение буфера с результатом ZwQuerySystemInformation.
                      الحصول على buffer بنتائج الدالة ZwQuerySystemInformation
*/
PVOID GetInfoTable(ULONG ATableType)
{
	ULONG mSize = 0x4000;
	PVOID mPtr = NULL;
	NTSTATUS St;
	do
	{
		mPtr = ExAllocatePool(PagedPool, mSize);
		memset(mPtr, 0, mSize);
		if (mPtr)
		{
			St = ZwQuerySystemInformation(ATableType, mPtr, mSize,
NULL);
		} else return NULL;
		if (St == STATUS_INFO_LENGTH_MISMATCH)
		{
			ExFreePool(mPtr);
			mSize = mSize * 2;
		}
	} while (St == STATUS_INFO_LENGTH_MISMATCH);
	if (St == STATUS_SUCCESS) return mPtr;
	ExFreePool(mPtr);
	return NULL;
}
```

اعتقد ان فهم الشفرة السابقة لن يمثل صعوبة لاي احد

الحصول على قائمة الprocesses من الحلقة الحاوية على الEPROCESS strctutes

الخطوة الثانية هي الحصول على الprocesses عن طريق المرور بجميع عناصر الحلقة والتي راسها هو PsActiveProcessHead لذلك من اجل الحصول على نتيجة سليمة لابد من ايجاد الراس. لتحقيق هذا الامر لابد من الاستفادة من كون الprocess بالاسم system هو اول process في قائمة الprocesses . بوقوعنا داخل DriverEntry فاننا نحتاج الى الحصول على مؤشر على الprocess current عن طريق PsGetCurrentProcess وبالازاحة سيشير الblink الى PsActiveProcessHead

```
PsActiveProcessHead = *(PVOID *)((PUCHAR)PsGetCurrentProcess +
ActiveProcessLinksOffset + 4);
```

الان يمكن المرور بجميع عناصر القائمة الثنائية الحلقية وانشاء قائمة بالprocesses

```
PVOID GetEprocessProcessList(ULONG *MemSize)
{
	PLIST_ENTRY Process;
	ULONG PsCount = 0;
	PVOID Mem = NULL;
	PProcessRecord Data;

	if (!PsActiveProcessHead) return NULL;

	Process = PsActiveProcessHead->Flink;

	while (Process != PsActiveProcessHead)
	{
		PsCount++;
		Process = Process->Flink;
```

```
        }

        PsCount++;

        *MemSize = PsCount * sizeof(TProcessRecord);

        Mem = ExAllocatePool(PagedPool, *MemSize);
        memset(Mem, 0, *MemSize);

        if (!Mem) return NULL; else Data = Mem;

        Process = PsActiveProcessHead->Flink;

        while (Process != PsActiveProcessHead)
        {
                Data->Present     = TRUE;
                Data->ProcessId   = *(PULONG)((ULONG)Process - ActPsLink +
pIdOffset);
                Data->ParrentPID  = *(PULONG)((ULONG)Process - ActPsLink +
ppIdOffset);
                Data->SignalState = *(PULONG)((ULONG)Process - ActPsLink + 4);
                Data->pEPROCESS   = (PEPROCESS)((ULONG)Process - ActPsLink);
                strncpy(Data->ProcessName, (PVOID)((ULONG)Process - ActPsLink +
NameOffset), 16);
                Data++;
            Process = Process->Flink;

        }

        return Mem;
}
```

للحصول على اسم الprocess والId والParrentProcessId نستخدم الازاحة داخل الحقول في الEPROCESS
(pIdOffset, ppIdOffset, NameOffset, ActPsLink)
طبعا مقدار الازاحة يختف من نسخة الى اخرى في الويندوز لذلك فان عملية الحصول على هذه القيم موضوع في دالة مستقلة بامكانك ان ترى هذه الاشياء في البرنامج الملحق المقال

طبعا فان اي latent process عن طريق التقاط الAPI سيتم كشفه بالطريقة السابقة لكن اذا كان الprocess مخفيا عن باسلوب الDKOM (Direct Kernel Object Manipulation) فان هذا الطريقة لن تعمل لان الprocess ينفصل عن الحلقة

الحصول على قائمة بالprocesses عن قوائم الthreads في الScheduler

احد اهم الطرق المستخدمة في العثور على الlatent processes هي الحصول على قائمة بالprocesses بواسطة قائمة الthreads في الScheduler. مثلا في windowos 2000 توجد 3 قوائم ثنائية للthreads :
KiWaitInListHead, KiWaitOutListHead, KiDispatcherReadyListHead

الاولان يستخدمان لتخزين (Expecting thread) للاحداث اما الاخير فهو للthreads الجاهزة للتنفيذ. بمرورنا بهذه القوائم و معرفة ازاحة قائمة الthread في الstructure ETHREAD سوف نحصل على مؤشر على الthread EThread. هذه الstructure تحوي على العديد من المؤشرات على المتصلة بالthread

struct _KPROCESS *Process (0x44, 0x150) and struct _EPROCESS *ThreadsProcess (0x22C, Displacement only for Windows 2000)
اول اثنين منهما لا يؤثران على عمل الthread لذلك من السهل تغييرهما لغرض السرية اما يستخدم من قبل الScheduler لاغراض التنقل بين الAddress spaces لذلك لا يمكن تغييره لذلك سوف نيتخدمه لمعرفة الprocess المالك للthread

هذه الطريقة مستخدمة في المنتج klister الذي يعمل فقط في win2000 ومع ذلك ليس مع النسخ.في الحقيقة ان تطعيم البرنامج بالعناوين (وضعها بشكل ستاتيكي) امر سيء جدا لان

هذا يضمن فشل البرنامج في العمل خصوصا بعد اي تحديث بل ويساعد على اختفاء الprocesses عن اعين ابرنامج
وهذا هو سبب فشل المنتج klister الذي وضع الaddresses lists في البرنامج بشكل ستاتيكي . بينما كان من الافضل حسابها بشكل ديناميكي بتحليل الاكواد الfunction التي تستخدم فيها هذه القوائم

كتجربة سنحاول ايجاد ال.KiWaitItListHead and KiWaitOutListHead in Windows 2000

عناوين هذه القوائم تستخدم في الدالة KeWaitForSingleObject في الكود التالي:

```
.text:0042DE56                  mov     ecx, offset KiWaitInListHead
.text:0042DE5B                  test    al, al
.text:0042DE5D                  jz      short loc_42DE6E
.text:0042DE5F                  cmp     byte ptr [esi+135h], 0
.text:0042DE66                  jz      short loc_42DE6E
.text:0042DE68                  cmp     byte ptr [esi+33h], 19h
.text:0042DE6C                  jl      short loc_42DE73
.text:0042DE6E                  mov     ecx, offset KiWaitOutListHead
```

للحصول على عناوين القوائم سنستخدم disassembler lengths instructions ولیكن LDasm اثناء تنقلنا داخل كود الدالة السابقة عندما يكون المؤشر pOpcode على السطر :

```
mov     ecx, offset KiWaitInListHead
```

فان العنوان pOpcode + 5 سيشير الى

```
test    al, al
```

pOpcode + 24 a سشير الى

```
mov     ecx, offset KiWaitOutListHead
```

فان عناوين KiWaitInListHead و KiWaitOutListHead سيوافقان المؤشرين pOpcode + 1 و pOpcode + 25 على حدة
سيكون البحث عن هذين العنوانين كالتالي :

```
void Win2KGetKiWaitInOutListHeads()
{
        PUCHAR cPtr, pOpcode;
        ULONG Length;

        for (cPtr = (PUCHAR)KeWaitForSingleObject;
             cPtr < (PUCHAR)KeWaitForSingleObject + PAGE_SIZE;
             cPtr += Length)
        {
                Length = SizeOfCode(cPtr, &pOpcode);

                if (!Length) break;

                if (*pOpcode == 0xB9 && *(pOpcode + 5) == 0x84 && *(pOpcode +
24) == 0xB9)
                {
                        KiWaitInListHead  = *(PLIST_ENTRY *)(pOpcode + 1);
                        KiWaitOutListHead = *(PLIST_ENTRY *)(pOpcode + 25);
                        break;
                }
        }

        return;
}
```

كذلك فان كود KiDispatcherReadyListHead في widowos2000 يكون بنفس الطريقة من طريق البحث في الدالة KeSetAffinityThread :

```
text:0042FAAA                  lea     eax, KiDispatcherReadyListHead[ecx*8]
```

```
.text:0042FAB1                  cmp     [eax], eax
```

هذه هي الدالة التي تبحث عن KiDispatcherReadyListHead :

```
void Win2KGetKiDispatcherReadyListHead()
{
        PUCHAR cPtr, pOpcode;
        ULONG Length;

        for (cPtr = (PUCHAR)KeSetAffinityThread;
             cPtr < (PUCHAR)KeSetAffinityThread + PAGE_SIZE;
             cPtr += Length)
        {
                Length = SizeOfCode(cPtr, &pOpcode);

                if (!Length) break;

                if (*(PUSHORT)pOpcode == 0x048D && *(pOpcode + 2) == 0xCD &&
*(pOpcode + 7) == 0x39)
                {
                        KiDispatcherReadyListHead = *(PVOID *)(pOpcode + 3);
                        break;
                }
        }

        return;
}
```

للاسف في winXP فان النواة تختلف اختلافا جذريا عن win2000 لذلك فان scheduler فقط قائمتين KiDispatcherReadyListHead and KiWaitListHead
بامكاننا نجد الKiWaitListHead في كود الدالة KeDelayExecutionThread :

```
.text:004055B5                  mov     dword ptr [ebx], offset KiWaitListHead
.text:004055BB                  mov     [ebx+4], eax
```

سننفذ البحث بهذه الطريقة :

```
{
        PUCHAR cPtr, pOpcode;
        ULONG Length;

        for (cPtr = (PUCHAR)KeDelayExecutionThread;
             cPtr < (PUCHAR)KeDelayExecutionThread + PAGE_SIZE;
             cPtr += Length)
        {
                Length = SizeOfCode(cPtr, &pOpcode);

                if (!Length) break;

                if (*(PUSHORT)cPtr == 0x03C7 && *(PUSHORT)(pOpcode + 6) ==
0x4389)
                {
                        KiWaitInListHead = *(PLIST_ENTRY *)(pOpcode + 2);
                        break;
                }
        }
```

اصعب شي هو ايجاد ال.KiDispatcherReadyListHead وذلك لان عنوان ال KiDispatcherReadyListHead ليس موجودا ولا في داله من exported functions لذلك للحصول عليه سنضطر الى تعقيد خوارزمية البحث

سنبدا البحث من الدالة KiDispatchInterrupt فيها يهمنا فقط مكان واحد وهو النكان الذي يحتوي علي الكود التالي:

```
text:00404E72                mov     byte ptr [edi+50h], 1
.text:00404E76               call    sub_404C5A
.text:00404E7B               mov     cl, 1
.text:00404E7D               call    sub_404EB9
```

اول call في هذه القطعة تشير الى الدالة التي تحتوي على اشارة الى KiDispatcherReadyListHead لكن البحث يصعب بسبب كون الدوال التي تشير اليه تملك تركيبا مختلفا في كل من winXp SP1,SP2 مثلا في SP2 فان هذه الدالة تملك هذا الشكل

```
.text:00404CCD               add     eax, 60h
.text:00404CD0               test    bl, bl
.text:00404CD2               lea     edx, KiDispatcherReadyListHead[ecx*8]
.text:00404CD9               jnz     loc_401F12
.text:00404CDF               mov     esi, [edx+4]
```

اما في SP1:

```
text:004180FE                add     eax, 60h
.text:00418101               cmp     [ebp+var_1], bl
.text:00418104               lea     edx, KiDispatcherReadyListHead[ecx*8]
.text:0041810B               jz      loc_418760
.text:00418111               mov     esi, [edx]
```

ان البحث فقط عن الامر lea ليس فعالا تماما لذلك سوف نقوم بمراقبة وجود اوامر بازاحة rel32 بعد الامر lea وهذا هو الكود الكامل للبحث عن KiDispatcherReadyListHead

```
void XPGetKiDispatcherReadyListHead()
{
        PUCHAR cPtr, pOpcode;
        PUCHAR CallAddr = NULL;
        ULONG Length;

        for (cPtr = (PUCHAR)KiDispatchInterrupt;
             cPtr < (PUCHAR)KiDispatchInterrupt + PAGE_SIZE;
             cPtr += Length)
        {
                Length = SizeOfCode(cPtr, &pOpcode);

                if (!Length) return;

                if (*pOpcode == 0xE8 && *(PUSHORT)(pOpcode + 5) == 0x01B1)
                {
                        CallAddr = (PUCHAR)(*(PULONG)(pOpcode + 1) + (ULONG)cPtr
+ Length);
                        break;
                }
        }

        if (!CallAddr || !MmIsAddressValid(CallAddr)) return;

        for (cPtr = CallAddr; cPtr < CallAddr + PAGE_SIZE; cPtr += Length)
        {
                Length = SizeOfCode(cPtr, &pOpcode);

                if (!Length) return;

                if (*(PUSHORT)pOpcode == 0x148D && *(pOpcode + 2) == 0xCD &&
IsRelativeCmd(pOpcode + 7))
                {
                        KiDispatcherReadyListHead = *(PLIST_ENTRY *)(pOpcode +
3);
                        break;
                }
        }
```

```
        return;
}
```

После нахождения адресов списков потоков, мы можем легко перечислить их процессы с помощью следующей функции:

```
void ProcessListHead(PLIST_ENTRY ListHead)
{
        PLIST_ENTRY Item;

        if (ListHead)
        {
                Item = ListHead->Flink;

                while (Item != ListHead)
                {
                        CollectProcess(*(PEPROCESS *)((ULONG)Item +
WaitProcOffset));
                        Item = Item->Flink;
                }
        }

        return;
}
```

CollectProcess هي دالة تقوم باضافة الprocess الى القائمة اذا لم يكن موجودافيها.

العثور على الprocesses عن طريق الsystem call

ان اي process يتعامل مع النظام عن طريق الAPI والعديد من طلبات الprocess تتحول الى طلب موجه الى نواة النظام طبعا بامكان الprocess ان يعمل دون اللجوء الى النظام ولكن عندها لن يستطيع عمل اي شيء نافع او ضار ولن يستطيع تنفيذ اي عمل . تكمن فكرة الطريقة الحاليه في التقاط اي طلب موجه الى System interface والحصول على مؤشر الى EPROCESS للPROCESS الحالي. قائمة المؤشرات لن تحتوي على ال processes التي لم تعمل اي Query اثناء عمل البرنامج بما في ذلك الاوامر التي كانت في حالة انتظار
اثناء عمل البرنامج.

في windows 2000 لعمل system call نستخدم المقاطعة 2EH لذلك كي نلتقط الsystem call سنحتاج الى تغيير مقبس 2EH الى idt لاجل هذا نحتاج الى معرفة وضع idt بواسطة الامر sidt هذا الامر سيعيد structure بهذا التركيب:

```
typedef struct _Idt
{
        USHORT Size;
        ULONG  Base;
} TIdt;
```

الكود الذي سيغير المقاطعة 2eh يبدو بالشكل التالي:

```
void Set2kSyscallHook()
{
        TIdt Idt;
        __asm
        {
                pushad
                cli
                sidt [Idt]
```

```
                mov esi, NewSyscall
                mov ebx, Idt.Base
                xchg [ebx + 0x170], si
                rol esi, 0x10
                xchg [ebx + 0x176], si
                ror esi, 0x10
                mov OldSyscall, esi
                sti
                popad
        }
}
```

طبعا قبل البدءلابد من تهيئة الوضع

```
void Win2kSyscallUnhook()
{
        TIdt Idt;
        __asm
        {
                pushad
                cli
                sidt [Idt]
                mov esi, OldSyscall
                mov ebx, Idt.Base
                mov [ebx + 0x170], si
                rol esi, 0x10
                mov [ebx + 0x176], si
                sti
                xor eax, eax
                mov OldSyscall, eax
                popad
        }
}
```

في winXP يستخدم system call interface قائم على اساس الاوامر sysenter/sysexit
الذين ظهرا في المعالج Pentium 2 .تتحكم (MSR) The model - specific registers بعمل
هاتين التعليمتين. يقع عنوان معالج الsystem call في رجستر الMSR :
SYSENTER_EIP_MSR(No 0x176) والقراءة من هذا الريجستر تنفذ بواسطة التعليمة rdmsr
قبل ذلك في ECX يجب ان يحوي رقم الرجستر الذي سوف نقرا منه اما نتيجة القراءه
فستكون في EDX:EAX في حالتنا هذه الرجستر SYSENTER_EIP_MSR عبارة عن 32-بت رجستر
لذا فان dx سيحوي على 0 بينما الAX سيحتوي على عنوان معالج الsystem call بالمثل
ممكن تنفيذ الكتابة داخل الMSR registers مع وجود فارق هو وجوب تصفير الEDX ولا فان
هذا يسبب
Exception وبالتالي انهيار مباشر للنظام
باخذ بعين الاعتبار ما قلناه سابقا فان الكود الذي سغير معالج system call

```
void SetXpSyscallHook()
{
        __asm
        {
                pushad
                mov ecx, 0x176
                rdmsr
                mov OldSyscall, eax
                mov eax, NewSyscall
                xor edx, edx
                wrmsr
                popad
        }
}
```

استعادة المعالج القديم

```
void XpSyscallUnhook()
{
        __asm
        {
                pushad
                mov ecx, 0x176
                mov eax, OldSyscall
                xor edx, edx
                wrmsr
                xor eax, eax
                mov OldSyscall, eax
                popad
        }
}
```

من خواص الXP انه يستطيع حداث الsystem call بكلا الطرقتين اعني بالsysenter وكذلك بالint 2Eh
لذلك سنضظر الى استبدال كلبهما ان المعالج الجديد يجب ان يحصل على مؤشر على EPROCESS للprocess الحالي واذا كان process جديد فلابد من اضافته الى القائمة
يبدو المعالج الجديد للsystem calls بهذا الشكل

```
void __declspec(naked) NewSyscall()
{
        __asm
        {
                pushad
                pushfd
                push fs
                mov di, 0x30
                mov fs, di
                mov eax, fs:[0x124]
                mov eax, [eax + 0x44]
                push eax
                call CollectProcess
                pop fs
                popfd
                popad
                jmp OldSyscall
        }
}
```

للحصول على قائمة بكل الprocesses لابد من ترك الكود السابق يعمل بعض الوقت. تبعا لعمل الكود سوف تواجهنا مشكلة :اذا انهي process معين اثناء وجوده في القائمة فان النتائج التي تلي هذه العملية ستكون خاطئة وسنوجد الprocess latennt خاطئ وقد نحل على BSOD . للخروج من هذا المأزق سوف نلجا الى الداله:
PsSetCreateProcessNotifyRoutine Callback
التي سيتم استدعائها عند انشاء او انهاء اي process لذلك عند انهاء عمل اي process سنحذفه من القائمة وهذا هو ال prototype الخاص بالcallback function

```
VOID
(*PCREATE_PROCESS_NOTIFY_ROUTINE) (
    IN HANDLE  ParentId,
    IN HANDLE  ProcessId,
    IN BOOLEAN  Create
    );
```

وسيتم تنصيب المعالج بهذاالشكل

```
PsSetCreateProcessNotifyRoutine(NotifyRoutine, FALSE);
```

وحذفه بهذا الشكل

```
PsSetCreateProcessNotifyRoutine(NotifyRoutine, TRUE);
```

ومع ذلك هناك مشكلة لا تخطر على بال مباشرة. داله الcallback دائما تستدعى في سياق انهاء الprocess لذلك من غير الممكن انهاء الprocess داخلها لذلك سنلجا الى احد الworker threads التابعة للنظام :اولا سنحجز ذاكرة لهذا الthread بواسطة ال IoAllocateWorkItem بعد ذلك نضع مهمتنا في الطابور التابع للThread بواسطة IoQueueWorkItem . اما داخل المعالج نفسه فلن نكتفي بحذف الprocess المنتهي فحسب بل وسنضيف الprocess المنشأ ايضا وهذا هو الكود الخاص بمالمعالج

```
void WorkItemProc(PDEVICE_OBJECT DeviceObject, PWorkItemStruct Data)
{
        KeWaitForSingleObject(Data->pEPROCESS, Executive, KernelMode, FALSE,
NULL);

        DelItem(&wLastItem, Data->pEPROCESS);

        ObDereferenceObject(Data->pEPROCESS);

        IoFreeWorkItem(Data->IoWorkItem);

        ExFreePool(Data);

        return;
}


void NotifyRoutine(IN HANDLE  ParentId,
                   IN HANDLE  ProcessId,
                   IN BOOLEAN Create)
{
        PEPROCESS       process;
        PWorkItemStruct Data;

        if (Create)
        {
                PsLookupProcessByProcessId(ProcessId, &process);

                if (!IsAdded(wLastItem, process)) AddItem(&wLastItem, process);

                ObDereferenceObject(process);

        } else
        {
                process = PsGetCurrentProcess();

                ObReferenceObject(process);

                Data = ExAllocatePool(NonPagedPool, sizeof(TWorkItemStruct));

                Data->IoWorkItem = IoAllocateWorkItem(deviceObject);

                Data->pEPROCESS  = process;

                IoQueueWorkItem(Data->IoWorkItem, WorkItemProc,
DelayedWorkQueue, Data);
        }

        return;
}
```

يعتبر هذا الاسلوب فعالا جدا في ايجاد الprocesses وذلك لاننا بدون ال system calling لن يستطيع المروراي process لكن بعض هذه الprocess قد يبقى وقتا طويلا في دون ان يعمل ايsystem call وهذا النوع من الprocesses لن نتمكن من تشخيصه ابدا.

ان تجاوز هذا اللاسلوب في العثور على latent processes ممكم ايضا وذلك عن طريق تغيير اسلوب تنفيذ الsystem call في الprocesses المراد اخفائها.

تغييره الى مقاطعة اخرى او الى callgate وخصوصا في XP حيث يكفي عمل patch لل KiFastSystemCall في ntdll.dll وانشاء الgateway المناسب للsystem call في win2000 اصعب نوعا بما ان الاستدعاء الint 2eh منتشرة في ال في ntdll لكن ايجاد كل وعمل patch لكل هذه الاماكن التي تحويه ليس صعبا لذلك لايجوز الاعتماد بشكل نهائي على
هذا الاختبار

الحصول قائمة بالprocesses بالاعتماد قائمة بجدول الhandles

اذا حدث وان اخفيت process معين عن طريق حذفه من قائمة الPsActiveProcesses لابد انك لاحظت انه تجده وتجد PID الخاص به عند سرد الhandles بواسطة الدالة ZwQuerySystemInformation ان الhandles التابعة له تشارك في القائمة ! يحدث هذا بسبب التالي: لجعل عملية سرد الhandles اسهل فان جميع جداول الhandles توضع في linked list ثنائي. ازاحةهذه القائمة في الhandle table تساوي 0x054 في win2000 Windows XP - 0x01C . ان الstructure HANDLE_TABLE تحتوي على مؤشر على الذي تبعه QuotaProcess ازاحة هذا المؤشر تساوي 0x00C في win 2000 وتساوي 0x004 في XP بالمرور بقائمة جداول الhandles سنستطيع تكوين قائمة بالprocesses في البداية نحتاج الى الراس (HandleTableListHead). لن نقوي بعمل disassembly كما فعلنا سابقا لان التجارب اثبتت انه يقع عميقا في النواة لكتننا سنستفيد من احد خواصه الاوهي كون الHandleTableListHead عباة عن Global variable ويعني انها تقع في احد اجزاء PE file التابع للنواة منها نستنتج اننا يجب ان تحصل على مؤشر على handletable اي عنصر ومن ثم والتحرك داخل linked list الى يظهر لنا العنصر الذي نشير اليه يقع داخل الPE file التبع للنواة وهذا العنصر سيكون handleTableListHead لايجاد حجم القاعدة وحجم ملف النواة نستخدم الدالة ZwQuerySystemInformation مع الصنف SystemModuleInformation . تحدد هذه الدالة مصفوفة بمواصفات ال loaded modules اول هناصر هذه المصفوفة هي النواة بمراعاة كل ما قيل سابقا فان كود البحث عن ال
HandleTableListHead سيكون بالشكل التالي:

```
void GetHandleTableListHead()
{
	PSYSTEM_MODULE_INFORMATION_EX Info =
GetInfoTable(SystemModuleInformation);
	ULONG NtoskrnlBase = (ULONG)Info->Modules[0].Base;
	ULONG NtoskrnlSize = Info->Modules[0].Size;
	PHANDLE_TABLE HandleTable = *(PHANDLE_TABLE
*)((ULONG)PsGetCurrentProcess() + HandleTableOffset);
	PLIST_ENTRY HandleTableList = (PLIST_ENTRY)((ULONG)HandleTable +
HandleTableListOffset);
	PLIST_ENTRY CurrTable;

	ExFreePool(Info);

	for (CurrTable = HandleTableList->Flink;
	     CurrTable != HandleTableList;
	     CurrTable = CurrTable->Flink)
	{
		if ((ULONG)CurrTable > NtoskrnlBase && (ULONG)CurrTable <
NtoskrnlBase + NtoskrnlSize)
		{
			HandleTableListHead = CurrTable;
			break;
		}
	}
}
```

هذا الكود عام جدا فهو يعنل لكل اصدارت الNT كما يمكن استخدامه ليس فقط مع ال HandleTableListHead وامكا مع كل القوائم التي تملك بنية مشابهة

بعد الحصول على HandleTableListHead يمكن ان نتحرك داخل القائمة الاساسية لجدوال الhandles ومن ثم انشاء قوائمة الprocesses بواسطتها.

```
void ScanHandleTablesList()
{
        PLIST_ENTRY CurrTable;
        PEPROCESS QuotaProcess;

        for (CurrTable =  HandleTableListHead->Flink;
             CurrTable != HandleTableListHead;
             CurrTable =  CurrTable->Flink)
        {
                QuotaProcess = *(PEPROCESS *)((PUCHAR)CurrTable - 
HandleTableListOffset + QuotaProcessOffset);
                if (QuotaProcess) CollectProcess(QuotaProcess);
        }
}
```

هذا الاسلوب في ايجاد الlatent processes يستخدم في البرنامج F-Secure Black Light وفي الاصادرالاخير من KProcCheck كيف يمكن تجاوزه اعتقد انكم تخمنون .

الحصول على قائمة بالحصول بالproccsses عن طريق عمل PspCidTable scanning

احد المفارقات عند اخفاء الprocess عن طريق حذفه من الPsActiveProcesses هي كون ذلك لا يعيق فتح الprocess بواسطة openProcess على هذا يقوم مبدا العثور على الprocesses عن طريق اختبار الpid ومحاولة فتح تلك الprocesses لن اقوم بتقديم مثال على هذه الطريقة التي اعتبرها سيئة وخالية من اي ميزة ولكن مع ذلك وجود هذه الطريقة تنبئ هن وجود قائمة اخرى يمكن بواسطتها فتح الprocesses الى جانب PsActiveProcesses كما ان الprocess الواحد يمكن ان يفتح باكثر من PID مما يوحي ان القائمة الاخرة التي نبحث عنها ماهي الا HANDLE_TABLE للتاكد من هذا الامر تعالوا ننظر الى الدالة ZwOpenProces :

```
PAGE:0049D59E ; NTSTATUS __stdcall NtOpenProcess(PHANDLE ProcessHandle, 
ACCESS_MASK DesiredAccess,
                                              POBJECT_ATTRIBUTES 
ObjectAttributes,PCLIENT_ID ClientId)
PAGE:0049D59E                   public NtOpenProcess
PAGE:0049D59E NtOpenProcess     proc near
PAGE:0049D59E
PAGE:0049D59E ProcessHandle     = dword ptr  4
PAGE:0049D59E DesiredAccess     = dword ptr  8
PAGE:0049D59E ObjectAttributes= dword ptr  0Ch
PAGE:0049D59E ClientId          = dword ptr  10h
PAGE:0049D59E
PAGE:0049D59E                   push    0C4h
PAGE:0049D5A3                   push    offset dword_413560 ; int
PAGE:0049D5A8                   call    sub_40BA92
PAGE:0049D5AD                   xor     esi, esi
PAGE:0049D5AF                   mov     [ebp-2Ch], esi
PAGE:0049D5B2                   xor     eax, eax
PAGE:0049D5B4                   lea     edi, [ebp-28h]
PAGE:0049D5B7                   stosd
PAGE:0049D5B8                   mov     eax, large fs:124h
PAGE:0049D5BE                   mov     al, [eax+140h]
PAGE:0049D5C4                   mov     [ebp-34h], al
PAGE:0049D5C7                   test    al, al
PAGE:0049D5C9                   jz      loc_4BE034
PAGE:0049D5CF                   mov     [ebp-4], esi
PAGE:0049D5D2                   mov     eax, MmUserProbeAddress
PAGE:0049D5D7                   mov     ecx, [ebp+8]
```

```
PAGE:0049D5DA                 cmp     ecx, eax
PAGE:0049D5DC                 jnb     loc_520CDE
PAGE:0049D5E2 loc_49D5E2:
PAGE:0049D5E2                 mov     eax, [ecx]
PAGE:0049D5E4                 mov     [ecx], eax
PAGE:0049D5E6                 mov     ebx, [ebp+10h]
PAGE:0049D5E9                 test    bl, 3
PAGE:0049D5EC                 jnz     loc_520CE5
PAGE:0049D5F2 loc_49D5F2:
PAGE:0049D5F2                 mov     eax, MmUserProbeAddress
PAGE:0049D5F7                 cmp     ebx, eax
PAGE:0049D5F9                 jnb     loc_520CEF
PAGE:0049D5FF loc_49D5FF:
PAGE:0049D5FF                 cmp     [ebx+8], esi
PAGE:0049D602                 setnz   byte ptr [ebp-1Ah]
PAGE:0049D606                 mov     ecx, [ebx+0Ch]
PAGE:0049D609                 mov     [ebp-38h], ecx
PAGE:0049D60C                 mov     ecx, [ebp+14h]
PAGE:0049D60F                 cmp     ecx, esi
PAGE:0049D611                 jz      loc_4CCB88
PAGE:0049D617                 test    cl, 3
PAGE:0049D61A                 jnz     loc_520CFB
PAGE:0049D620 loc_49D620:
PAGE:0049D620                 cmp     ecx, eax
PAGE:0049D622                 jnb     loc_520D0D
PAGE:0049D628 loc_49D628:
PAGE:0049D628                 mov     eax, [ecx]
PAGE:0049D62A                 mov     [ebp-2Ch], eax
PAGE:0049D62D                 mov     eax, [ecx+4]
PAGE:0049D630                 mov     [ebp-28h], eax
PAGE:0049D633                 mov     byte ptr [ebp-19h], 1
PAGE:0049D637 loc_49D637:
PAGE:0049D637                 or      dword ptr [ebp-4], 0FFFFFFFFh
PAGE:0049D63B loc_49D63B:
PAGE:0049D63B
PAGE:0049D63B                 cmp     byte ptr [ebp-1Ah], 0
PAGE:0049D63F                 jnz     loc_520D34
PAGE:0049D645 loc_49D645:
PAGE:0049D645                 mov     eax, PsProcessType
PAGE:0049D64A                 add     eax, 68h
PAGE:0049D64D                 push    eax
PAGE:0049D64E                 push    dword ptr [ebp+0Ch]
PAGE:0049D651                 lea     eax, [ebp-0D4h]
PAGE:0049D657                 push    eax
PAGE:0049D658                 lea     eax, [ebp-0B8h]
PAGE:0049D65E                 push    eax
PAGE:0049D65F                 call    SeCreateAccessState
PAGE:0049D664                 cmp     eax, esi
PAGE:0049D666                 jl      loc_49D718
PAGE:0049D66C                 push    dword ptr [ebp-34h] ; PreviousMode
PAGE:0049D66F                 push    ds:stru_5B6978.HighPart
PAGE:0049D675                 push    ds:stru_5B6978.LowPart ; PrivilegeValue
PAGE:0049D67B                 call    SeSinglePrivilegeCheck
PAGE:0049D680                 test    al, al
PAGE:0049D682                 jnz     loc_4AA7DB
PAGE:0049D688 loc_49D688:
PAGE:0049D688                 cmp     byte ptr [ebp-1Ah], 0
PAGE:0049D68C                 jnz     loc_520D52
PAGE:0049D692                 cmp     byte ptr [ebp-19h], 0
PAGE:0049D696                 jz      loc_4CCB9A
PAGE:0049D69C                 mov     [ebp-30h], esi
PAGE:0049D69F                 cmp     [ebp-28h], esi
PAGE:0049D6A2                 jnz     loc_4C1301
PAGE:0049D6A8                 lea     eax, [ebp-24h]
PAGE:0049D6AB                 push    eax
```

```
PAGE:0049D6AC                  push    dword ptr [ebp-2Ch]
PAGE:0049D6AF                  call    PsLookupProcessByProcessId
PAGE:0049D6B4 loc_49D6B4:
```

كما ترون ان الكود السابق وبطريقة امنه يقوم بنسخ القيم المنقولة ويراقب وجودها على حدود عناوين المستخدم يراقب وجود الAccess rights و جود privilege "SeDebugPrivilege " وبعد ذلك ياخذ الProcessId من الCLIENT_ID structure ويرسلها الى الداله PsLookupProcessByProcessId والتي مهمتها هي الحصول على مؤشر على EPROCESS بواسطة الProcessId با قي اجزاء الدالة السابقة ليس مهما لذلك سنلقي على الدالة PsLookupProcessByProcessId

```
PAGE:0049D725                  public PsLookupProcessByProcessId
PAGE:0049D725 PsLookupProcessByProcessId proc near
PAGE:0049D725
PAGE:0049D725
PAGE:0049D725 ProcessId        = dword ptr  8
PAGE:0049D725 Process          = dword ptr  0Ch
PAGE:0049D725
PAGE:0049D725                  mov     edi, edi
PAGE:0049D727                  push    ebp
PAGE:0049D728                  mov     ebp, esp
PAGE:0049D72A                  push    ebx
PAGE:0049D72B                  push    esi
PAGE:0049D72C                  mov     eax, large fs:124h
PAGE:0049D732                  push    [ebp+ProcessId]
PAGE:0049D735                  mov     esi, eax
PAGE:0049D737                  dec     dword ptr [esi+0D4h]
PAGE:0049D73D                  push    PspCidTable
PAGE:0049D743                  call    ExMapHandleToPointer
PAGE:0049D748                  mov     ebx, eax
PAGE:0049D74A                  test    ebx, ebx
PAGE:0049D74C                  mov     [ebp+ProcessId], STATUS_INVALID_PARAMETER
PAGE:0049D753                  jz      short loc_49D787
PAGE:0049D755                  push    edi
PAGE:0049D756                  mov     edi, [ebx]
PAGE:0049D758                  cmp     byte ptr [edi], 3
PAGE:0049D75B                  jnz     short loc_49D77A
PAGE:0049D75D                  cmp     dword ptr [edi+1A4h], 0
PAGE:0049D764                  jz      short loc_49D77A
PAGE:0049D766                  mov     ecx, edi
PAGE:0049D768                  call    sub_4134A9
PAGE:0049D76D                  test    al, al
PAGE:0049D76F                  jz      short loc_49D77A
PAGE:0049D771                  mov     eax, [ebp+Process]
PAGE:0049D774                  and     [ebp+ProcessId], 0
PAGE:0049D778                  mov     [eax], edi
PAGE:0049D77A loc_49D77A:
PAGE:0049D77A                  push    ebx
PAGE:0049D77B                  push    PspCidTable
PAGE:0049D781                  call    ExUnlockHandleTableEntry
PAGE:0049D786                  pop     edi
PAGE:0049D787 loc_49D787:
PAGE:0049D787                  inc     dword ptr [esi+0D4h]
PAGE:0049D78D                  jnz     short loc_49D79A
PAGE:0049D78F                  lea     eax, [esi+34h]
PAGE:0049D792                  cmp     [eax], eax
PAGE:0049D794                  jnz     loc_52388A
PAGE:0049D79A loc_49D79A:
PAGE:0049D79A                  mov     eax, [ebp+ProcessId]
PAGE:0049D79D                  pop     esi
PAGE:0049D79E                  pop     ebx
PAGE:0049D79F                  pop     ebp
PAGE:0049D7A0                  retn    8
```

ان ما ترونه يثبت وبما لايقطع الشك يوجود قائمة اخرى على لـHANDLE_TABLE اسم القائمة او الجدول واسمها هو PspCidTable وتخزن داخلها قائمة جدول باسماء الprocesses والthreads وتستخدم هذه القائمة في ايضا في الدوال PsLookupProcessThreadByCid and PsLookupThreadByThreadId. كما ترون فان الhandle مؤشر على جدول الhandles يرسلان الى الدالة ExMapHandleToPointer والتي تعيد بدورها مؤشرا على عنصر من عناصر الجدول بحيث هذا العنصر بوصف الhandle المرسل :

```
struct _HANDLE_TABLE_ENTRY {
 // static data ------------------------------------
 // non-static data --------------------------------
  /*<thisrel this+0x0>*/ /*|0x4|*/ void* Object;
  /*<thisrel this+0x0>*/ /*|0x4|*/ unsigned long ObAttributes;
  /*<thisrel this+0x0>*/ /*|0x4|*/ struct _HANDLE_TABLE_ENTRY_INFO* InfoTable;
  /*<thisrel this+0x0>*/ /*|0x4|*/ unsigned long Value;
  /*<thisrel this+0x4>*/ /*|0x4|*/ unsigned long GrantedAccess;
  /*<thisrel this+0x4>*/ /*|0x2|*/ unsigned short GrantedAccessIndex;
  /*<thisrel this+0x6>*/ /*|0x2|*/ unsigned short CreatorBackTraceIndex;
  /*<thisrel this+0x4>*/ /*|0x4|*/ long NextFreeTableEntry;
  };// <size 0x8>
```

ومنه يمكن ان نحصل على التركيب التالي لـHANDLE_TABLE_ENTRY :

```
typedef struct _HANDLE_TABLE_ENTRY
{
    union
    {
        PVOID                     Object;
        ULONG                     ObAttributes;
        PHANDLE_TABLE_ENTRY_INFO InfoTable;
        ULONG                     Value;
    };

    union
   {
       union
       {
          ACCESS_MASK GrantedAccess;
          struct
          {
              USHORT GrantedAccessIndex;
              USHORT CreatorBackTraceIndex;
          };
       };

       LONG NextFreeTableEntry;
   };
} HANDLE_TABLE_ENTRY, *PHANDLE_TABLE_ENTRY;
```

ماهو الشي المفيد الذي نستطيع الحصول عليه ؟ اولا محتوى الحقل object والذي يعتبر مجموع المؤشرات على الhandle الموصوف و الflag الذي يشير الى كون العنصر من الجدول مشغولا ام لا؟ مثلا من برنامج اخر(لاحقا سنعرف اهمية هذا الشي.) كما ان هناك الحقل البالغ الاهمية GrantedAccess الذي يبين الحقوق المسموح بها في التعامل مع الobject .مثلا امكانية فتح ملف للقراة ثم محاولة الكتابة داخله.الان لابد ان نفهم صيغ جداول الhandles لكي نتمكن من سردها في قائمة. هنا تبدا ملامح اختلاف قوي بين الwin2000 ال XP وسنضطر الى التعامل مع مر واحدة منهما على حدة. في البداية سنتناول صيغة الجدول في win2000 فهي اسهل للفهم وسننظر الى الان كود الدالة :

ExMapHandleToPointer:

```
PAGE:00493285 ExMapHandleToPointer proc near
PAGE:00493285
PAGE:00493285
PAGE:00493285 HandleTable     = dword ptr  8
PAGE:00493285 Handle          = dword ptr  0Ch
PAGE:00493285
```

```
PAGE:00493285                 push    esi
PAGE:00493286                 push    [esp+Handle]
PAGE:0049328A                 push    [esp+4+HandleTable]
PAGE:0049328E                 call    ExpLookupHandleTableEntry
PAGE:00493293                 mov     esi, eax
PAGE:00493295                 test    esi, esi
PAGE:00493297                 jz      short loc_4932A9
PAGE:00493299                 push    esi
PAGE:0049329A                 push    [esp+4+HandleTable]
PAGE:0049329E                 call    ExLockHandleTableEntry
PAGE:004932A3                 neg     al
PAGE:004932A5                 sbb     eax, eax
PAGE:004932A7                 and     eax, esi
PAGE:004932A9 loc_4932A9:
PAGE:004932A9                 pop     esi
PAGE:004932AA                 retn    8
PAGE:004932AA ExMapHandleToPointer endp
```

وهنا استدعاء للداله ExMapHandleToPointer والتب تحدث البحث في HANDLE_TABLE كما اننا نرى استدعاء الدالة ExLockHandleTableEntry التي تعمل lock bit installation ولفهم عمل الhandles سنضطر الى فهم عمل الدالتين وسنبدا ب ExpLookupHandleTableEntry:

وهنا استدعاء للدالة

```
PAGE:00493545 ExpLookupHandleTableEntry proc near
PAGE:00493545
PAGE:00493545
PAGE:00493545 HandleTable     = dword ptr  0Ch
PAGE:00493545 Handle          = dword ptr  10h
PAGE:00493545
PAGE:00493545                 push    esi
PAGE:00493546                 push    edi
PAGE:00493547                 mov     edi, [esp+Handle]
PAGE:0049354B                 mov     eax, 0FFh
PAGE:00493550                 mov     ecx, edi
PAGE:00493552                 mov     edx, edi
PAGE:00493554                 mov     esi, edi
PAGE:00493556                 shr     ecx, 12h
PAGE:00493559                 shr     edx, 0Ah
PAGE:0049355C                 shr     esi, 2
PAGE:0049355F                 and     ecx, eax
PAGE:00493561                 and     edx, eax
PAGE:00493563                 and     esi, eax
PAGE:00493565                 test    edi, 0FC000000h
PAGE:0049356B                 jnz     short loc_49358A
PAGE:0049356D                 mov     eax, [esp+HandleTable]
PAGE:00493571                 mov     eax, [eax+8]
PAGE:00493574                 mov     ecx, [eax+ecx*4]
PAGE:00493577                 test    ecx, ecx
PAGE:00493579                 jz      short loc_49358A
PAGE:0049357B                 mov     ecx, [ecx+edx*4]
PAGE:0049357E                 test    ecx, ecx
PAGE:00493580                 jz      short loc_49358A
PAGE:00493582                 lea     eax, [ecx+esi*8]
PAGE:00493585 loc_493585:
PAGE:00493585                 pop     edi
PAGE:00493586                 pop     esi
PAGE:00493587                 retn    8
PAGE:0049358A loc_49358A:
PAGE:0049358A                 xor     eax, eax
PAGE:0049358C                 jmp     short loc_493585
PAGE:0049358C ExpLookupHandleTableEntry endp
```

واضافة لهذا ارفق تركيب HANDLE_TABLE structure والتي حصلهنا من ntoskkrnl.pdb

```
struct _HANDLE_TABLE {
  // static data ------------------------------------
  // non-static data --------------------------------
  /*<thisrel this+0x0>*/ /*|0x4|*/ unsigned long Flags;
  /*<thisrel this+0x4>*/ /*|0x4|*/ long HandleCount;
  /*<thisrel this+0x8>*/ /*|0x4|*/ struct _HANDLE_TABLE_ENTRY*** Table;
  /*<thisrel this+0xc>*/ /*|0x4|*/ struct _EPROCESS* QuotaProcess;
  /*<thisrel this+0x10>*/ /*|0x4|*/ void* UniqueProcessId;
  /*<thisrel this+0x14>*/ /*|0x4|*/ long FirstFreeTableEntry;
  /*<thisrel this+0x18>*/ /*|0x4|*/ long NextIndexNeedingPool;
  /*<thisrel this+0x1c>*/ /*|0x38|*/ struct _ERESOURCE HandleTableLock;
  /*<thisrel this+0x54>*/ /*|0x8|*/ struct _LIST_ENTRY HandleTableList;
  /*<thisrel this+0x5C>*/ /*|0x10|*/ struct _KEVENT HandleContentionEvent;
}; // <size 0x6c>
```

وبهذا سنعيد بناء الhandles tables structure

```
typedef struct _WIN2K_HANDLE_TABLE
{
        ULONG                 Flags;
        LONG                  HandleCount;
        PHANDLE_TABLE_ENTRY **Table;
        PEPROCESS             QuotaProcess;
        HANDLE                UniqueProcessId;
        LONG                  FirstFreeTableEntry;
        LONG                  NextIndexNeedingPool;
        ERESOURCE             HandleTableLock;
        LIST_ENTRY            HandleTableList;
        KEVENT                HandleContentionEvent;
} WIN2K_HANDLE_TABLE , *PWIN2K_HANDLE_TABLE ;
```

من البديهي ان قيمة الhandle تتوزع الى 3 اجزاء التي تعبر عن indexes

في جدول الكائنات ذو المستويات الثلاث . والان سننظر الى كود الدالة

ExLockHandleTableEntry:

```
PAGE:00492E2B ExLockHandleTableEntry proc near
PAGE:00492E2B
PAGE:00492E2B
PAGE:00492E2B var_8           = dword ptr -8
PAGE:00492E2B var_4           = dword ptr -4
PAGE:00492E2B HandleTable     = dword ptr  8
PAGE:00492E2B Entry           = dword ptr  0Ch
PAGE:00492E2B
PAGE:00492E2B                 push    ebp
PAGE:00492E2C                 mov     ebp, esp
PAGE:00492E2E                 push    ecx
PAGE:00492E2F                 push    ecx
PAGE:00492E30                 push    ebx
PAGE:00492E31                 push    esi
PAGE:00492E32                 xor     ebx, ebx
PAGE:00492E34 loc_492E34:
PAGE:00492E34                 mov     eax, [ebp+Entry]
PAGE:00492E37                 mov     esi, [eax]
PAGE:00492E39                 test    esi, esi
PAGE:00492E3B                 mov     [ebp+var_8], esi
PAGE:00492E3E                 jz      short loc_492E89
```

```
PAGE:00492E40                 jle     short loc_492E64
PAGE:00492E42                 mov     eax, esi
PAGE:00492E44                 or      eax, 80000000h      // set
WIN2K_TABLE_ENTRY_LOCK_BIT
PAGE:00492E49                 mov     [ebp+var_4], eax
PAGE:00492E4C                 mov     eax, [ebp+var_8]
PAGE:00492E4F                 mov     ecx, [ebp+Entry]
PAGE:00492E52                 mov     edx, [ebp+var_4]
PAGE:00492E55                 cmpxchg [ecx], edx
PAGE:00492E58                 cmp     eax, esi
PAGE:00492E5A                 jnz     short loc_492E64
PAGE:00492E5C                 mov     al, 1
PAGE:00492E5E loc_492E5E:
PAGE:00492E5E                 pop     esi
PAGE:00492E5F                 pop     ebx
PAGE:00492E60                 leave
PAGE:00492E61                 retn    8
PAGE:00492E64 loc_492E64:
PAGE:00492E64                 mov     eax, ebx
PAGE:00492E66                 inc     ebx
PAGE:00492E67                 cmp     eax, 1
PAGE:00492E6A                 jb      loc_4BC234
PAGE:00492E70                 mov     eax, [ebp+HandleTable]
PAGE:00492E73                 push    offset unk_46D240 ; Timeout
PAGE:00492E78                 push    0               ; Alertable
PAGE:00492E7A                 push    0               ; WaitMode
PAGE:00492E7C                 add     eax, 5Ch
PAGE:00492E7F                 push    0               ; WaitReason
PAGE:00492E81                 push    eax             ; Object
PAGE:00492E82                 call    KeWaitForSingleObject
PAGE:00492E87                 jmp     short loc_492E34
PAGE:00492E89 loc_492E89:
PAGE:00492E89                 xor     al, al
PAGE:00492E8B                 jmp     short loc_492E5E
PAGE:00492E8B ExLockHandleTableEntry endp
```

يعمل الكود التالي على على فحص 31 بت في العنصر object من
الHANDLE_TABLE_ENTRY structrue ويمنحه القيمة 1 اما اذا كان البت يمتلك القيمة 1 (من دون ان نتدخل في منحه هذه القيمة) فسوف ينتظر HandleContentionEvent في الHANDLE_TABLE . ان ما يهمنا هو منح الTABLE_ENTRY_LOCK_BIT القيمة 1 لانه جز كم عنوان الobject واذا كانت قيمته 0 فاننا سنحصل على القيمة 1
اما تركيب الHANDLE_TABLE فاعتقد اننا تحدثنا عنه سابقا والان نستطيع ايراد الكود الذي يفعل ما تحدثنا عنه :

```
void ScanWin2KHandleTable(PWIN2K_HANDLE_TABLE HandleTable)
{
        int i, j, k;
        PHANDLE_TABLE_ENTRY Entry;

        for (i = 0; i < 0x100; i++)
        {
                if (HandleTable->Table[i])
                {
                        for (j = 0; j < 0x100; j++)
                        {
                                if (HandleTable->Table[i][j])
                                {
                                        for (k = 0; k < 0x100; k++)
                                        {
                                                Entry = &HandleTable-
>Table[i][j][k];

                                                if (Entry->Object)

ProcessObject((PVOID)((ULONG)Entry->Object | WIN2K_TABLE_ENTRY_LOCK_BIT));
```

```
                                        }
                                }
                        }
                }
        }
}
```

تقوم هذه الدالة بفحص جميع الobjects في الجدول ثم تستدعي دالة ProcessObject لمعرفة نوع الobject ومن ثم معالجته كما يجب وهذا هو كود :

```
void ProcessObject(PVOID Object)
{
        POBJECT_HEADER ObjectHeader = OBJECT_TO_OBJECT_HEADER(Object);

        if (ObjectHeader->Type == *PsProcessType) CollectProcess(Object);

        if (ObjectHeader->Type == *PsThreadType)  ThreadCollect(Object);
}
```

بذلك نكون قد انتهينا من جدول الكائنات الخاص بwin2000 الى سنقوم بعمل نفس الشي في XP

اولا سنقوم بعمل disassembly للدالةExpLookupHandleTableEntry :

```
PAGE:0048D3C1 ExpLookupHandleTableEntry proc near
PAGE:0048D3C1
PAGE:0048D3C1
PAGE:0048D3C1 HandleTable     = dword ptr  8
PAGE:0048D3C1 Handle          = dword ptr  0Ch
PAGE:0048D3C1
PAGE:0048D3C1                 mov     edi, edi
PAGE:0048D3C3                 push    ebp
PAGE:0048D3C4                 mov     ebp, esp
PAGE:0048D3C6                 and     [ebp+Handle], 0FFFFFFFCh
PAGE:0048D3CA                 mov     eax, [ebp+Handle]
PAGE:0048D3CD                 mov     ecx, [ebp+HandleTable]
PAGE:0048D3D0                 mov     edx, [ebp+Handle]
PAGE:0048D3D3                 shr     eax, 2
PAGE:0048D3D6                 cmp     edx, [ecx+38h]
PAGE:0048D3D9                 jnb     loc_4958D6
PAGE:0048D3DF                 push    esi
PAGE:0048D3E0                 mov     esi, [ecx]
PAGE:0048D3E2                 mov     ecx, esi
PAGE:0048D3E4                 and     ecx, 3     // ecx - table level
PAGE:0048D3E7                 and     esi, not 3 // esi - pointer to first table
PAGE:0048D3EA                 sub     ecx, 0
PAGE:0048D3ED                 jnz     loc_48DEA4
PAGE:0048D3F3                 lea     eax, [esi+eax*8]
PAGE:0048D3F6 loc_48D3F6:
PAGE:0048D3F6                 pop     esi
PAGE:0048D3F7 loc_48D3F7:
PAGE:0048D3F7                 pop     ebp
PAGE:0048D3F8                 retn    8
PAGE:0048DEA4 loc_48DEA4:
PAGE:0048DEA4                 dec     ecx
PAGE:0048DEA5                 mov     ecx, eax
PAGE:0048DEA7                 jnz     loc_52F57A
PAGE:0048DEAD                 shr     ecx, 9
PAGE:0048DEB0                 mov     ecx, [esi+ecx*4]
PAGE:0048DEB3 loc_48DEB3:
PAGE:0048DEB3                 and     eax, 1FFh
```

```
PAGE:0048DEB8                lea     eax, [ecx+eax*8]
PAGE:0048DEBB                jmp     loc_48D3F6
PAGE:0052F57A loc_52F57A:
PAGE:0052F57A                shr     ecx, 13h
PAGE:0052F57D                mov     edx, ecx
PAGE:0052F57F                mov     ecx, [esi+ecx*4]
PAGE:0052F582                shl     edx, 13h
PAGE:0052F585                sub     eax, edx
PAGE:0052F587                mov     edx, eax
PAGE:0052F589                shr     edx, 9
PAGE:0052F58C                mov     ecx, [ecx+edx*4]
PAGE:0052F58F                jmp     loc_48DEB3
```

والان سننشيء الHANDLE_TABLE structure من HANDLE_TABLE damp

```
struct _HANDLE_TABLE {
 // static data ------------------------------------
 // non-static data --------------------------------
  /*<thisrel this+0x0>*/ /*|0x4|*/ unsigned long TableCode;
  /*<thisrel this+0x4>*/ /*|0x4|*/ struct _EPROCESS* QuotaProcess;
  /*<thisrel this+0x8>*/ /*|0x4|*/ void* UniqueProcessId;
  /*<thisrel this+0xc>*/ /*|0x10|*/ struct _EX_PUSH_LOCK HandleTableLock[4];
  /*<thisrel this+0x1c>*/ /*|0x8|*/ struct _LIST_ENTRY HandleTableList;
  /*<thisrel this+0x24>*/ /*|0x4|*/ struct _EX_PUSH_LOCK HandleContentionEvent;
  /*<thisrel this+0x28>*/ /*|0x4|*/ struct _HANDLE_TRACE_DEBUG_INFO* DebugInfo;
  /*<thisrel this+0x2c>*/ /*|0x4|*/ long ExtraInfoPages;
  /*<thisrel this+0x30>*/ /*|0x4|*/ unsigned long FirstFree;
  /*<thisrel this+0x34>*/ /*|0x4|*/ unsigned long LastFree;
  /*<thisrel this+0x38>*/ /*|0x4|*/ unsigned long NextHandleNeedingPool;
  /*<thisrel this+0x3c>*/ /*|0x4|*/ long HandleCount;
  /*<thisrel this+0x40>*/ /*|0x4|*/ unsigned long Flags;
  /*<bitfield this+0x40>*/ /*|0x1|*/ unsigned char StrictFIFO:0:1;
  };  // <size 0x44>
```

وسنعيد بناء الجدول :

```
typedef struct _XP_HANDLE_TABLE
{
       ULONG                     TableCode;
       PEPROCESS                 QuotaProcess;
       PVOID                     UniqueProcessId;
       EX_PUSH_LOCK              HandleTableLock[4];
       LIST_ENTRY                HandleTableList;
       EX_PUSH_LOCK              HandleContentionEvent;
       PHANDLE_TRACE_DEBUG_INFO  DebugInfo;
       LONG                      ExtraInfoPages;
       ULONG                     FirstFree;
       ULONG                     LastFree;
       ULONG                     NextHandleNeedingPool;
       LONG                      HandleCount;
       LONG                      Flags;
       UCHAR                     StrictFIFO;
} XP_HANDLE_TABLE, *PXP_HANDLE_TABLE;
```

من الكود النذكور اعلاه واضح ان الدالة ExpLookupHandleTableEntry تستخرج قيمة TableCode من HANDLE_TABLE واعتمادا على اصغر 2 بت (ذو الرتبة الاقل) نحدد عدد مستويات الجدول اما البتات المتبقية فتشكل مؤشرا على جدول من المستوى الاول نستنتج ان HANDLE_TABLE في WinXp قد يمتلك من 1-3 مستويات وحجم الجدول في كل مستوى يساوي 1FFH منا ان النظام يستطيع زيادة عدد المستويات تلقائيا في حالو زيادة عدد السجلات في الجدول .من البدبهي ان المستوى الثاني يتكون فقط عن عندما يكون عدد السجلات في الجدول اكثر من 0x200 اما الثالث فقط عند 0x40000 ولادري اذا ماكان النظام يقلل هد المستويات في حالة تحرير عدد من سجلات الجدول او على الاقل لم الحظ ذلك

ان الدلة ExLockHandleTableEntry غيرموجودة في win Xp فام كود عنصرالجدول المحدث للblocking يقع في الدالة ExMapHandleToPointer وبعمل disassembly لهذه الدالة :

```
PAGE:0048F61E ExMapHandleToPointer proc near
PAGE:0048F61E
PAGE:0048F61E
PAGE:0048F61E var_8          = dword ptr -8
PAGE:0048F61E var_4          = dword ptr -4
PAGE:0048F61E HandleTable    = dword ptr  8
PAGE:0048F61E Handle         = dword ptr  0Ch
PAGE:0048F61E
PAGE:0048F61E                mov     edi, edi
PAGE:0048F620                push    ebp
PAGE:0048F621                mov     ebp, esp
PAGE:0048F623                push    ecx
PAGE:0048F624                push    ecx
PAGE:0048F625                push    edi
PAGE:0048F626                mov     edi, [ebp+Handle]
PAGE:0048F629                test    di, 7FCh
PAGE:0048F62E                jz      loc_4A2A36
PAGE:0048F634                push    ebx
PAGE:0048F635                push    esi
PAGE:0048F636                push    edi
PAGE:0048F637                push    [ebp+HandleTable]
PAGE:0048F63A                call    ExpLookupHandleTableEntry
PAGE:0048F63F                mov     esi, eax
PAGE:0048F641                test    esi, esi
PAGE:0048F643                jz      loc_4A2711
PAGE:0048F649                mov     [ebp+var_4], esi
PAGE:0048F64C loc_48F64C:
PAGE:0048F64C                mov     ebx, [esi]
PAGE:0048F64E                test    bl, 1
PAGE:0048F651                mov     [ebp+var_8], ebx
PAGE:0048F654                jz      loc_508844
PAGE:0048F65A                lea     eax, [ebx-1]
PAGE:0048F65D                mov     [ebp+Handle], eax
PAGE:0048F660                mov     eax, [ebp+var_8]
PAGE:0048F663                mov     ecx, [ebp+var_4]
PAGE:0048F666                mov     edx, [ebp+Handle]
PAGE:0048F669                cmpxchg [ecx], edx
PAGE:0048F66C                cmp     eax, ebx
PAGE:0048F66E                jnz     loc_50884C
PAGE:0048F674                mov     eax, esi
PAGE:0048F676 loc_48F676:
PAGE:0048F676                pop     esi
PAGE:0048F677                pop     ebx
PAGE:0048F678 loc_48F678:
PAGE:0048F678                pop     edi
PAGE:0048F679                leave
PAGE:0048F67A                retn    8
PAGE:0048F67A ExMapHandleToPointer endp
```

بعد ذلك ننتظر حتى تعيد الدالة ExpLookupHandleTableEntry مؤشرا على HANDLE_TABLE_ENTRY وسنقوم باختبار اصغر بت في الحقل object اذا كان يساوي 1 فاننا نمنحه القيمة 1 والا فاننا ننتظر منحه القيمة 1 نستنتج من هذا اننا لن الى منح 1 لاكبر بت القيمة 1 (كما في 2000) وانما نحتاج الى تصفير اصغر بت لمراعاة ما سباق سوف نكتب الكود الذي يعمل scan لجدول الكائنات:

```
void ScanXpHandleTable(PXP_HANDLE_TABLE HandleTable)
{
        int i, j, k;
        PHANDLE_TABLE_ENTRY Entry;
        ULONG TableCode = HandleTable->TableCode & ~TABLE_LEVEL_MASK;

        switch (HandleTable->TableCode & TABLE_LEVEL_MASK)
```

```
        {
               case 0 :
                 for (i = 0; i < 0x200; i++)
                 {
                        Entry = &((PHANDLE_TABLE_ENTRY)TableCode)[i];

                        if (Entry->Object) ProcessObject((PVOID)((ULONG)Entry->Object & ~XP_TABLE_ENTRY_LOCK_BIT));
                 }
               break;

               case 1 :
                 for (i = 0; i < 0x200; i++)
                 {
                        if (((PVOID *)TableCode)[i])
                        {
                               for (j = 0; j < 0x200; j++)
                               {
                                      Entry = &((PHANDLE_TABLE_ENTRY *)TableCode)[i][j];

                                      if (Entry->Object)
ProcessObject((PVOID)((ULONG)Entry->Object & ~XP_TABLE_ENTRY_LOCK_BIT));
                               }
                        }
                 }
               break;

               case 2 :
                 for (i = 0; i < 0x200; i++)
                 {
                        if (((PVOID *)TableCode)[i])
                        {
                               for (j = 0; j < 0x200; j++)
                               {
                                      if (((PVOID **)TableCode)[i][j])
                                      {
                                             for (k = 0; k < 0x200; k++)
                                             {
                                                    Entry =
&((PHANDLE_TABLE_ENTRY **)TableCode)[i][j][k];

                                                    if (Entry->Object)

ProcessObject((PVOID)((ULONG)Entry->Object & ~XP_TABLE_ENTRY_LOCK_BIT));
                                             }
                                      }
                               }
                        }
                 }
               break;
        }
}
```

والان فهمنا تركيب جدول الobjects لم يتبقى سرد الprocesses
(بصراحة من كثر الكلام نسيت اننا ندور الprocesses .. المترجم)
و لعمل ذلك نا نحتاج الى ايجاد عنوان PspCidTable وكما توقعتم فاننا سنجده عن طريق عمل disassembly للدالة PsLookupProcessByProcessId والتي فيها اول Call سيحوي عنوان PspCidTable وهذا هو الكود الذي يعمل ذلك :

```
void GetPspCidTable()
{
        PUCHAR cPtr, pOpcode;
```

```
        ULONG Length;

        for (cPtr = (PUCHAR)PsLookupProcessByProcessId;
             cPtr < (PUCHAR)PsLookupProcessByProcessId + PAGE_SIZE;
             cPtr += Length)
        {
              Length = SizeOfCode(cPtr, &pOpcode);

              if (!Length) break;

              if (*(PUSHORT)cPtr == 0x35FF && *(pOpcode + 6) == 0xE8)
              {
                      PspCidTable = **(PVOID **)(pOpcode + 2);
                      break;
              }
        }
}
```

اعتقد اننا انتهينا من PspCidTable ولا اعتقد ان هناك صعوبة لكل من فهم ما هو مكتوب اعلاه في انشاء برامج مماثلة.

الحصول على قائمة بالprocesses عن طريق التقاط SwapContext

اذا كان في النظام process ما نشطا فلا د انه يمتلك threads خاصة بها واذا تمكننامن التقاط عملية تنقل النظام بين الthreads فلابد اننا بواسطة هذه العملية نستطيع الحصول على معلومات تخص الprocesses التي تتبعها هذه الthreads. قبل ذلك لابد لنا ان نعرف عملية التنقل بين الthreads: بعد فترة زمنية محددة (10-15 ms) يقوم الsystem timre باستدعاء مقاطعة تحت الscheduler بحيث يقوم الاخير بالانتقال الى الthread التالي اذا كانت الفترة الزمنية الخاصة بالthread الحالي قد انتهت.

اما عملية الانتقال نفسها فانها تتم عن طريق Not exported function من النواة بالاسم SwapContex والتي يمكن ان تجدها في ntoskrnl.exe المهم ان هذه الدالة التي يستدعيها ال**scheduler عند انتهاء الفترة الزمنية (Quantum of time)** او عندما ينتظر ال thread الحالي معينا .في الحالة الاولى فان الدالة السابقة يتم استدعاءها من KiDispatchInterrupt اما في الحالة الثانية فانه يتم استدعاءها من **Not exported function واقعة في اعماق النواة والتي بدورها تستدعى من** KeWaitForSingleObject و KeDelayExecutionThread و KeWaitForMultipleObjects يتم ارسال بارمترات الدالة SwapContext الى الرجسترات وهي كالتالي :

Mode of processing APC - Cl
Edi – مؤشر على الthread المتروك
Esi – مؤشر على الthread الذي سننتقل اليه
Ebx – مؤشر على PCR .

ما يهمنا الان هو المؤشرات التي تخص الthreads التي يتم التنقل بينها في الرجسترين esi,edi. في بداية الامر ما يهمنا هو ايجاد عنوان الدالة SwapContext لذلك فاننا سنعمل Disassembly للدالة KiDispatchInterrupt وسوف نبحث عن كود بهذا الشكل

```
.text:00404E76                call     sub_404C5A
.text:00404E7B                mov      cl, 1
.text:00404E7D                call     SwapContext
```

ومنها نستطيع ان نستخرج عنوان الدالة. في الحقيقة ان هذه الطريقة في البحث فعالة جدا حيث يسمح لنا ان نجد الدالة swapContext في نسخدة من نسخ الwindows ابتداءا من 2000 وحتى 2003 وبكافة اللاصدارات وهذا هو الكود الذي سيقوم بعلمية البحث عن الدالة SwapContext :

```
void GetSwapContextAddress()
```

```
{
        PUCHAR cPtr, pOpcode;
        ULONG Length;

        for (cPtr = (PUCHAR)KiDispatchInterrupt;
             cPtr < (PUCHAR)KiDispatchInterrupt + PAGE_SIZE;
             cPtr += Length)
        {
                Length = SizeOfCode(cPtr, &pOpcode);

                if (!Length) break;

                if (*(PUSHORT)pOpcode == 0x01B1 && *(pOpcode + 2) == 0xE8)
                {
                        pSwapContext = (PVOID)(*(PULONG)(pOpcode + 3) +
(ULONG)cPtr + 7);
                        break;
                }
        }

        return;
}
```

الان وبعد ان وجدنا عنوان الدالة سنحتاج الى التقاطها (hook) الطريقة الوحيدة في هذه الحالة هو عمل splicing لعمل ذلك سنفعل التالي سننسخ بضع تعليمات من بداية كود الدالة الملتقطة يعد ذلك سنضع في اخره التعليمة jmp ثم في الدالة الخاصة بنا (والتي سنقفز اليها)
اثناء ذلك نبحث عن وجود
relative offset
ونقوم بتعديله والا سنحصل على BSOD .
لوضع واتنزاع تلك ال(Hooks) اليكم الكود :

```
#define MemOpen()  __asm cli; __asm mov eax, cr0; __asm mov oData, eax; \
                   __asm and eax, 0xFFFEFFFF; __asm mov cr0, eax;
#define MemClose() __asm mov eax, oData; __asm mov cr0, eax; __asm sti;


UCHAR SaveOldFunction(PUCHAR Proc, PUCHAR Old)
{
        ULONG  Size;
        PUCHAR pOpcode;
        ULONG  Offset;
        PUCHAR oPtr;
        ULONG Result = 0;

        Offset = (ULONG)Proc - (ULONG)Old;
        oPtr = Old;
        while (Result < 5)
        {
                Size = SizeOfCode(Proc, &pOpcode);
                memcpy(oPtr, Proc, Size);
                if (IsRelativeCmd(pOpcode))    *(PULONG)((ULONG)pOpcode -
(ULONG)Proc + (ULONG)oPtr + 1) += Offset;
                oPtr   += Size;
                Proc   += Size;
                Result += Size;
        }
        *(PUCHAR)((ULONG)Old + Result) = 0xE9;
        *(PULONG)((ULONG)Old + Result + 1) = Offset - 5;
        return (UCHAR)Result;
}
```

```
PVOID HookCode(PVOID TargetProc, PVOID NewProc)
{
        ULONG Address;
        PVOID OldFunction;
        PVOID Proc = TargetProc;
        ULONG oData;

        Address = (ULONG)NewProc - (ULONG)Proc - 5;
        MemOpen();
        OldFunction = ExAllocatePool(NonPagedPool, 20);
        *(PULONG)OldFunction = (ULONG)Proc;
        *(PUCHAR)((ULONG)OldFunction + 4) = SaveOldFunction((PUCHAR)Proc,
(PUCHAR)((ULONG)OldFunction + 5));
        *(PUCHAR)Proc = 0xE9;
        *(PULONG)((ULONG)Proc + 1) = Address;
        MemClose();
        return (PVOID)((ULONG)OldFunction + 5);
}


void UnhookCode(PVOID OldProc)
{
        PUCHAR Proc, pMem;
        PUCHAR pOpcode;
        ULONG  Size, ThisSize;
        ULONG  SaveSize, Offset;
        ULONG oData;

        Proc = (PUCHAR)(*(PULONG)((ULONG)OldProc - 5));
        pMem = Proc;
        SaveSize = *(PUCHAR)((ULONG)OldProc - 1);
        Offset   = (ULONG)Proc - (ULONG)OldProc;
        MemOpen();
        memcpy(Proc, OldProc, SaveSize);
        ThisSize = 0;
        while (ThisSize < SaveSize)
        {
                Size = SizeOfCode(Proc, &pOpcode);
                if (IsRelativeCmd(pOpcode)) *(PULONG)((ULONG)pOpcode + 1) -=
Offset;
                Proc     += Size;
                ThisSize += Size;
        }
        MemClose();
        ExFreePool((PVOID)((ULONG)OldProc - 5));
        return;
}
```

وهذا هو كود الدالة SwapContext البديل

```
void __declspec(naked) NewSwapContext()
{
        __asm
        {
                pushad
                pushfd
                push edi
                call ThreadCollect
                push esi
                call ThreadCollect
                popfd
                popad
                jmp OldSwapContext
        }
}
```

ان اسلوب splicing اكيد انه سهل ومريح لكنه خطيروذلك في لحظة وضع وانتزاع الhook لانه قد يسبب مشاكل كثيرة وخصوصا في الانظمة متعددة الprocesses (او مع Hiperthreading processor) حيث يستطيه thread اخر استدعاو الدالة التي التي التقاطناها في الوقت الذي ينتهي الpatch من عمله تعالوا لنحسب احتمال وقوع اضرار:
لنفترض ان patch بداية الدالة يستغرق ms 0.01 (طبعا الزمن بالميكروثانية..المترجم)(في الحقيقة اقل من ذلك بكثير) عندئذ سكون احتمال انهيار النظام 0,00000785عبارة عن مرة واحدة لكل 127380 مرة تشغيل.ان مثل هذا الاحتمال لبرامج مثل RootKit Detector مقبولة جدا اذا اخذنا بعبن الا عتبار ان مثل هذه البرامج نادرا ما تعمل اساسا.
لمن ماذاعن تبك البرامج التي تعمل باستمرار مثلا مضادات الفيروسات فان استخدام الhooks يسمح به فقط مرة فقط عن بدء لقلاع نظام التشغيل في مثل هذه الحظة احتمال انهيارالنظان صغير جدا لدرجة اننا نستطيع اهماله .

كل ماذا ذكرته سابقا موجود في البرنامج المرفق مع المقال. طبعا تجاوز كل ما سبق من طرق العثور على الlatent processes عملية صعبة للغاية لكن صانعي الroot hooks يتجاوزن ذلك عن طرق عمل التقاط IOCTL من البرنامج الى الدرابفر مبا شرة. افضل طريقة من هذه النوع هو استخدان نوع سري من الdetectors